Frame-based Elastic Models

BENJAMIN GILLES
University of British Columbia, Vancouver, CANADA
and
GUILLAUME BOUSQUET
Grenoble Universités, INRIA, LJK-CNRS, FRANCE
and
FRANCOIS FAURE
Grenoble Universités, INRIA, LJK-CNRS, FRANCE
and
DINESH K. PAI
University of British Columbia, Vancouver, CANADA

We present a new type of deformable model which combines the realism of physically based continuum mechanics models and the usability of frame-based skinning methods. The degrees of freedom are coordinate frames. In contrast with traditional skinning, frame positions are not scripted but move in reaction to internal body forces. The displacement field is smoothly interpolated using dual quaternion blending. The deformation gradient and its derivatives are computed at each sample point of a deformed object and used in the equations of Lagrangian mechanics to achieve physical realism. This allows easy and very intuitive definition of the degrees of freedom of the deformable object. The meshless discretization allows on-the-fly insertion of frames to create local deformations where needed. We formulate the dynamics of these models in detail and describe some pre-computations that can be used for speed. We show that our method is effective for behaviors ranging from simple unimodal deformations to complex realistic deformations comparable with Finite Element simulations. To encourage its use, the software will be freely available in the simulation platform SOFA.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically-Based Modeling*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: algorithms

Additional Key Words and Phrases: Physically based animation, deformable solids

DOI
http://doi.acm.org/

1. INTRODUCTION

Deformable models are essential for computer animation, especially for animating characters and soft objects. In current practice, however, the animator has to choose between two very different approaches (see Sec. 2 for a brief review).

One approach is *skinning* (also known as vertex blending or skeletal subspace deformation). The deformation is kinematically generated by manipulating "bones," i.e., specific coordinate frames. This method is widely used, not only for its simplicity and efficiency, but because it provides natural and intuitive handles for controlling deformation. Skinning generates smooth deformations using a very sparse sampling of the deformation field. Adaptation is simple since frames can be inserted easily to control local features. These interesting features have made it the most widely used method for character animation. However, as a consequence of its purely kinematic nature (i.e., the frame positions need to be scripted), achieving physically realistic dynamic deformation is a major challenge with this approach.

The other approach is *physically based deformation*, typically using continuum mechanics. This has the significant advantage that physical realism is "baked in" right from the start. Complex animations are generated by numerical integration of discretized differential equations. However, these methods can be expensive and difficult to use. In the popular Finite Element Method (FEM) framework, the degrees of freedom of the discretized model are the vertices of a mesh, which must be constructed for each simulation object. A relatively fine mesh (i.e., a dense sampling of the deformation field) is required to capture common deformations such as torsion, leading to expensive simulations. Mesh adaptation can be difficult due to the topological constraints of the mesh. Particle-based meshless methods have been proposed to address these problems. While they obviate the need to maintain mesh topology, particles can not be placed arbitrarily, since each material point has to be in the range of at least four non-coplanar particles. Therefore, these methods also need a dense cloud of particles not very different from the vertices of an FEM mesh.

In this paper, we propose a new approach that combines the advantages of both skinning and physically based deformation. Instead of the vertices of a mesh, the degrees of freedom are a sparse set of coordinate frames, which parameterize material points using an advanced skinning method called dual quaternion blending

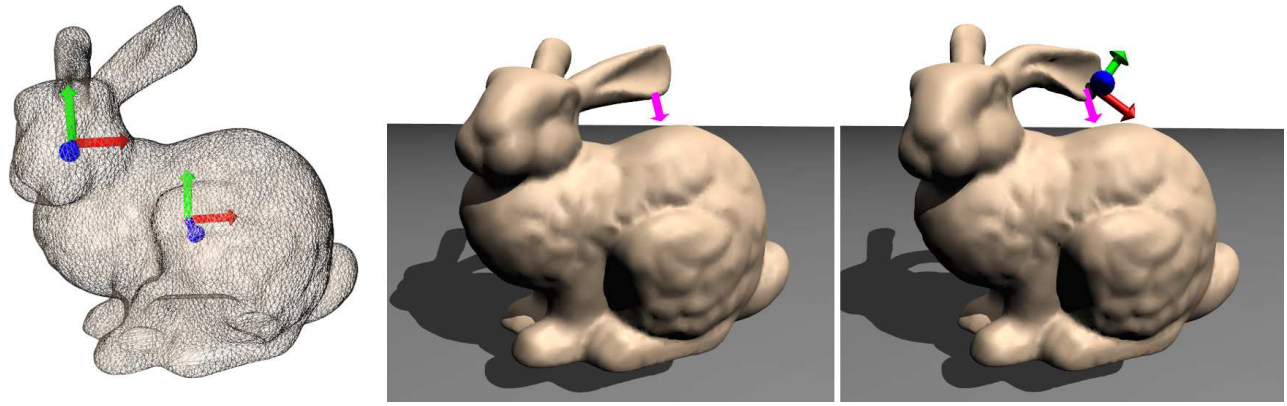

Fig. 1. Frame-based model features. Physical realism with very sparse sampling: two frames are sufficient to model a dynamically deformable bunny (Left). Intuitive animator control: The head and the ears are almost rigid (Middle), but by inserting an additional frame (Right), the ear becomes flexible.

[Kavan et al. 2008]. The equations of motion are derived for the moving frames by applying the principles of continuum mechanics across the volume of the deformed object, and solved using classical implicit time integration.

Contributions

Our main contribution is a new approach which unifies skinning and physically based deformation modeling. This allows the creation of efficient physically sound models with very sparse and intuitive sampling, and on-the-fly adaptation to create local deformations where needed.

In contrast with FEM and traditional particle-based meshless methods, there is no constraint on the number of frames influencing each point of the deformable object, because a single frame is sufficient to represent a displacement field including rotations. This also allows very sparse sampling; indeed, some simple deformable models are easily modeled using a couple of frames.

Complex materials can be accurately modeled using maps of distributed material properties such as density and stiffness. The weight functions of the control frames also provide the animator with an additional, very intuitive, method to model the physical behavior required in an animation. For instance, rigid regions are easily obtained by locally associating a large weight to a given frame and small weights to the others. Moreover, we show that it is possible to encode the mass and stiffness matrices in six-dimensional mass-spring systems which are computationally efficient and allow large deformations. Finally, local deformations can be adaptively modeled, using dynamically inserted control frames with appropriate weight functions.

The remainder of the paper is organized as follows. Previous work is summarized in Section 2. The physics of our deformable model is presented in Section 3. Application to object modeling is presented in Section 4. Results are discussed in Section 5 and conclusions are drawn in Section 6.

2. BACKGROUND

Physically based deformable models have attracted continuous attention in Computer Graphics, since the seminal work of Terzopoulos [Terzopoulos et al. 1987]. We refer the reader to the excellent survey of [Nealen et al. 2005] on this topic. Here, we briefly review the main Lagrangian models of deformable objects, followed by a short introduction to skinning.

Mesh-based methods. Early works on deformable models in Computer Graphics have focused on interconnected particles. In mass-spring systems [Platt and Badler 1981], constraints on edge length are enforced to counter stretching. Bending and shear can be controlled using additional springs. More general constraints such as area or volume conservation can be enforced using appropriate energy functions [Teschner et al. 2004]. To realistically model volumetric deformable objects, it is necessary to apply continuum mechanics. The spatial derivatives of the displacement field can be computed using finite differences on a regular grid [Terzopoulos et al. 1987]. [Terzopoulos and Qin 1994] studied the case of physically deformable NURBS surfaces for shape modeling. Finite elements [Bathe 1996; Gourret et al. 1989; O'Brien and Hodgins 1999; Cotin et al. 1999] allow irregular meshes, which are generally more convenient to sample objects with arbitrary shapes, but may be poorly conditioned. The spatial domain is subdivided into elements such as triangles, hexahedra or more frequently tetrahedra, in which the displacement field is interpolated using shape functions. At each point the strain can be computed using the spatial derivatives of the displacement field. Accurate material models have been implemented from rheological models relating stress and strain in hyperelastic, viscoelastic, inhomogeneous, transversely isotropic and/or quasi-incompressible media [Weiss et al. 1996]. For simplicity, linearized strain has been applied assuming small displacements in rotated frames [Müller and Gross 2004]. Precomputed deformations modes have been used to interactively deform large structures [Barbič and James 2005]. Models based on Cosserat points have been proposed for large deformations in thin structures [Pai 2002] and solids [Nadler and Rubin 2003]. Since robustness problems such as inverted tetrahedra [Irving et al. 2006] or hourglass deformation modes in hexahedra [Nadler and Rubin 2003] have been addressed, meshing remain the main issue in finite elements. To reduce computation time, embedding detailed objects in coarse meshes has become popular in computer graphics [Müller and Gross 2004; Sifakis et al. 2007; Nesme et al. 2009]. Multi-resolution approaches have been proposed [Debunne et al. 2001; Grinspun et al. 2002]. In recent work, disconnected or arbitrarily-shaped ele-

ments [Kaufmann et al. 2008; Martin et al. 2008] have been proposed to alleviate the meshing difficulties.

Meshless methods. Meshless methods do not use an underlying embedding structure but unstructured control points. In computer graphics, meshless methods have been first introduced for fluid simulation and then extended to solid mechanics [Müller et al. 2004; Gross and Pfister 2007]. Each control point has a given influence that generally decreases with the distance to it. Standard approximation or interpolation methods have been investigated for physical simulation such as Shepard functions, radial basis functions and moving least squares (see [Fries and Matthies 2003] for an extensive review). Despite the added flexibility due to the absence of elements, the approximation function still requires a certain overlap between the influence regions of control points. Particularly the approximation of rotations requires at least four control points, contrary to our method that explicitly use rotations in the degrees of freedom. [Martin et al. 2010] alleviate this problem using an extended moving least squares method based on deformable frames instead of particles. Their main focus is on a novel differential measure of deformation, called elaston, combined with standard RBF weight functions and dense sampling, resulting in large computation times. In contrast, our focus is on fast computation and ease of modeling, using sparse sampling and customized shape functions.

Besides continuum mechanics-based methods, fast algorithms have been developed for video games to simulate quasi-isometry [Adams et al. 2008; Müller et al. 2005]. They are not able to model real materials, being based on geometry only.

Skinning. Meshless approximation functions estimate a continuous deformation field at locations influenced by at least four control points while finite element functions interpolate deformation inside elements. A third approach, unexploited in physical simulation until now, consists in interpolating rigid transformations directly. In character animation, this is well known as *skinning* (or vertex blending or skeletal subspace deformation), in which soft-tissues are deformed by blending rigid transforms [Magnenat-Thalmann et al. 1988]. Various rigid motion interpolation methods have been proposed and we refer the reader to [Kavan et al. 2008] for a detailed review. Specifically, *dual quaternion blending* offers a good approximation of the linear interpolation of screws at a reasonable computational cost [Kavan et al. 2007]. It provides a closed-form solution for more than two transforms contrary to screw interpolation that requires an iterative treatment. We believe that this makes it well suited for parameterizing a physically based deformable model; we exploit these features in our model.

Basics and notation. We now provide background on dual quaternion skinning and introduce the notations used in our derivation of the corresponding 3d elasticity method. Let $\bar{\mathbf{p}} = [\bar{x}\,\bar{y}\,\bar{z}]^T$ be the initial coordinates of a material point in the world coordinates system, as illustrated in figure 2. To simplify the derivations and without loss of generality, we assume that $[\bar{x}\,\bar{y}\,\bar{z}]$ are also the coordinates of the point in the local parameterization of the object. In this paper, we use a horizontal bar on top to denote the value in the reference, undeformed, position. Using skinning, the displaced position $\mathbf{p} = [x\,y\,z]^T$ of the point is computed as

$$\mathbf{p} = \mathbf{R}\bar{\mathbf{p}} + \mathbf{t} \tag{1}$$

where the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ are computed using a weighted sum of the displacements of one or several moving frames influencing the point. In linear blend skinning, $\mathbf{R}$ and $\mathbf{t}$ are simple weighted sums of each frame's matrix and vector, and well-known artifacts occur with large rotations. These problems can be alleviated using dual quaternion skinning. Dual quaternions are pairs of quaternions with a special algebra which represent rigid transforms when normalized. Most of quaternion properties hold for dual quaternions. We briefly outline the method here, and refer the reader to [Kavan et al. 2007; Kavan et al. 2008] for more details. The position of frame i is represented using a dual quaternion written as a 8d vector $\mathbf{q}_i = [\mathbf{q}_0^{iT}\,\mathbf{q}_\varepsilon^{iT}]^T$ where $\mathbf{q}_0^i$ is a unit quaternion representing rotation and $\mathbf{q}_\varepsilon^i = (\mathbf{t}_{i\wedge} \circ \mathbf{q}_0^i)/2$ is a quaternion representing translation to the frame origin $\mathbf{t}_i$. Operator $\circ$ denotes standard quaternion product and $\wedge$ denotes conversion from a 3d vector to a quaternion with null scalar part. Blended displacements are computed as normalized weighted sums of dual quaternions:

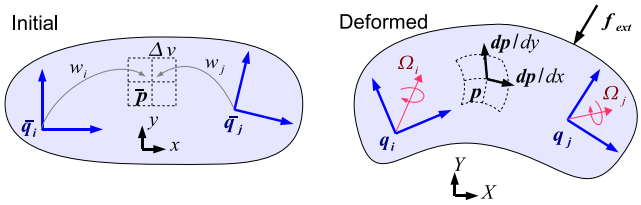

Fig. 2. An object undergoing skinning deformation using two moving frames.

$$\mathbf{b}' = \frac{\mathbf{b}}{\|\mathbf{b}\|} = \frac{\sum w_i \bar{\mathbf{T}}_i \mathbf{q}_i}{\|\sum w_i \bar{\mathbf{T}}_i \mathbf{q}_i\|} \tag{2}$$

where the 8×8 matrix $\bar{\mathbf{T}}_i$ encodes dual quaternion product with $\bar{\mathbf{q}}_i^{-1}$, the inverse of the frame transform in the reference position. This product represents the transformation applied to the reference configuration, and it is used as a kinematic parameter in [Kavan et al. 2008]. The matrix and vector of equation (1) are then straightforwardly deduced from $\mathbf{b}'$ (see appendix A). Figure 2 shows some isolines of the parameterization, depicted as initially orthogonal lines within a volume element Δv centered on point $\bar{\mathbf{p}}$. The weights w_i are functions of $\bar{\mathbf{p}}$, so the blended displacement $\{\mathbf{R}, \mathbf{t}\}$ must be computed at each vertex of the deformed geometry. We represent frame velocity using the 6×1 vector $\mathbf{\Omega}_i = [\boldsymbol{\omega}_i^T\,\dot{\mathbf{t}}_i^T]^T$ where $\dot{\mathbf{t}}_i$ and $\boldsymbol{\omega}_i$ are the linear and angular velocities. Similar to standard quaternions, there is a linear relationship between the velocity vector and the rate of the frame [Han et al. 2008]: $\dot{\mathbf{q}}_i = (\hat{\boldsymbol{\omega}}_i \circ \mathbf{q}_i)/2$, where $\hat{\boldsymbol{\omega}}_i = [\boldsymbol{\omega}_{i\wedge}^T\,(\dot{\mathbf{t}}_i + \mathbf{t}_i \times \boldsymbol{\omega}_i)_\wedge^T]^T$ is a dual quaternion representing rigid body velocity. We encode this linear relationship using the 8×6 matrix $\mathbf{L}_i$ such as $\dot{\mathbf{q}}_i = \mathbf{L}_i \mathbf{\Omega}_i$. We introduce the directional derivative operator $\boldsymbol{\nabla}_i$ to denote differentiation with respect to the six independent degrees of freedom of frame i , *i.e.* for a given N-dimensional vector $\mathbf{u}(\mathbf{q})$, we have $\dot{\mathbf{u}} = \sum_i \boldsymbol{\nabla}_i \mathbf{u}\, \mathbf{\Omega}_i$ where $\boldsymbol{\nabla}_i \mathbf{u}$ is a $N \times 6$ matrix.

3. THE DYNAMICS OF FRAME-BASED CONTINUUM

As shown in the following diagram, we apply a classical methodology in continuum mechanics: from the degrees of freedom (i.e., frame positions and orientations), we interpolate the displacement field in the material (i.e., dual quaternion skinning, equation (2)), from which we compute the strain and strain derivatives through spatial differentiation (section 3.3). Material properties are then used to compute the internal elastic energy from the strain (section 3.4). By differentiation with respect to the DOFs, we obtain,

for each frame, a generalized force and force Jacobian (section 3.5) that we integrate in time to compute the acceleration, velocity and the new position. The simulation loop is summarized in the following diagram; the symbols are defined in the next section.

Position		Strain		Energy
$\mathbf{q}$	$\xrightarrow{\partial}$	$\mathbf{E}$	$\xrightarrow{\text{Material}}$	W
$\int \uparrow$				$\downarrow \boldsymbol{\nabla}$
$\boldsymbol{\Omega}$	$\xleftarrow{\int}$	$\dot{\boldsymbol{\Omega}}$	$\xleftarrow{\text{Mass}}$	$\boldsymbol{\Gamma}$
Velocity		Acceleration		Force

In the remainder of this section, we first set up the dynamics equation, then we derive the local terms of this equation for a continuum animated using dual quaternion skinning.

3.1 Differential Equation

Models derived using Lagrangian mechanics are ordinary differential equations which have the following form in generalized coordinates:

$$\mathbf{M}\dot{\boldsymbol{\Omega}} - \boldsymbol{\Gamma}(\mathbf{q}, \boldsymbol{\Omega}) = \boldsymbol{\Gamma}_{\boldsymbol{ext}}(\mathbf{q}, \boldsymbol{\Omega}) \tag{3}$$

Here $\mathbf{q}$ and $\boldsymbol{\Omega}$ are the positions and velocities described in the previous section, $\dot{\boldsymbol{\Omega}}$ denotes accelerations, $\boldsymbol{\Gamma}$ internal forces, $\boldsymbol{\Gamma}_{\boldsymbol{ext}}$ external and inertial forces, and $\mathbf{M}$ is the mass matrix. Numerous methods have been proposed to solve this ODE, and without loss of generality we will use Implicit Euler integration (e.g., [Baraff and Witkin 1998]), which computes velocity updates by solving the following equation:

$$\left(\mathbf{M} - h\mathbf{C} - h^2\mathbf{K}\right)\delta\boldsymbol{\Omega} = h\left(\boldsymbol{\Gamma}_{\boldsymbol{ext}} + h\mathbf{K}\boldsymbol{\Omega}\right) \tag{4}$$

where h is the time step, $\mathbf{K} = \frac{d\boldsymbol{\Gamma}}{d\mathbf{q}}$ is the stiffness matrix, and $\mathbf{C} = \frac{d\boldsymbol{\Gamma}}{d\boldsymbol{\Omega}}$ the damping matrix. In this paper, we use the popular Rayleigh assumption: $\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K}$.

Frame positions are updated based on the updated velocities:

$$\begin{aligned} \mathbf{q}_0^i(t+h) &= \mathbf{q}_0^i(t) + \frac{h}{2}(\boldsymbol{\omega}_{i\wedge} \circ \mathbf{q}_0^i(t)) \\ \mathbf{t}_i(t+h) &= \mathbf{t}_i(t) + h\dot{\mathbf{t}}_i \ , \end{aligned} \tag{5}$$

followed by a re-normalization of the quaternion part.

3.2 Mass

The generalized mass matrix can be computed by assembling 6×6 blocks:

$$\mathbf{M}_{ij} = \int_{\mathcal{V}} \rho \mathbf{J}_i^T \mathbf{J}_j d\mathcal{V} \tag{6}$$

where ρ is the mass density, and the 3×6 Jacobian $\mathbf{J}_i$ maps a frame velocity vector to a point velocity such as: $\dot{\mathbf{p}} = \sum_i \mathbf{J}_i \boldsymbol{\Omega}_i$. The Jacobian is obtained using chain rule differentiation of equation (1):

$$\mathbf{J}_i = \boldsymbol{\nabla}_i \mathbf{p} = \frac{\partial \mathbf{p}}{\partial \mathbf{b}'}\frac{\partial \mathbf{b}'}{\partial \mathbf{b}}\frac{\partial \mathbf{b}}{\partial \mathbf{q}_i}\boldsymbol{\nabla}_i \mathbf{q}_i = w_i \mathbf{Q}\mathbf{N}\bar{\mathbf{T}}_i\mathbf{L}_i \tag{7}$$

Matrix $\bar{\mathbf{T}}_{i\,(8\times8)}$ and $\mathbf{L}_{i\,(8\times6)}$ were introduced in section 2. Matrix $\mathbf{N}_{(8\times8)}$ is a projection which filters out the changes of norm of the dual quaternion. Matrix $\mathbf{Q}_{(3\times8)}$ encodes displacement of the given point associated to a change of the normalized dual quaternion $\mathbf{b}'$. The expressions of these matrices are given in appendix A.

3.3 Strain

In the rest shape, the columns of the deformation gradient $\mathbf{dp}/\mathbf{d\bar{p}}$ form an orthonormal reference frame, while in deformed state *stretching* corresponds to vector length change, and *shearing* corresponds to angles, as illustrated in figure 2. The deformation gradient can be computed by differentiating equation (1):

$$\mathbf{F} = \frac{\mathbf{dp}(\bar{\mathbf{p}}, \mathbf{b}'(\bar{\mathbf{p}}))}{\mathbf{d\bar{p}}} = \frac{\partial \mathbf{p}}{\partial \bar{\mathbf{p}}} + \frac{\partial \mathbf{p}}{\partial \mathbf{b}'}\frac{\partial \mathbf{b}'}{\partial \mathbf{b}}\frac{\partial \mathbf{b}}{\partial \bar{\mathbf{p}}} = \mathbf{R} + \mathbf{QNW} \tag{8}$$

Matrix $\mathbf{W}_{(8\times3)}$ encodes the dependence of the weighted sum $\mathbf{b}$ on the material point and depends on the local variations $\mathbf{W}_i = \partial w_i/\partial \bar{\mathbf{p}}$ of the weight functions in the material (see appendix A).

Three dimensional strain measures are generally based on the deformation gradient. Due to space restrictions, in the remainder of this paper we focus on the popular Green-Lagrange strain tensor, valid for large displacements. The six independent terms of this tensor $\boldsymbol{E} = (\mathbf{F}^\mathbf{T}\mathbf{F} - \mathbf{I})/2$ can be represented in vector form as: $\mathbf{E} = [\epsilon_{xx}\ \epsilon_{yy}\ \epsilon_{zz}\ \epsilon_{xy}\ \epsilon_{yz}\ \epsilon_{zx}]^T$.

3.4 Strain energy and stress

The strain energy density $W(\bar{\mathbf{p}})$, representing the potential elastic energy per unit volume stored at a given point, is: $W = \mathbf{E}^\mathbf{T}\mathbf{S}/2$ where $\mathbf{S}$ is the stress in a vector form. Due to space restrictions, we focus on Hookean elasticity and obtain the popular St. Venant-Kirchhoff material: $\mathbf{S} = \mathbf{HE}$, where $\mathbf{H}$ is the standard 6×6 Hooke's stiffness matrix. The density of strain energy is thus $W = \mathbf{E}^\mathbf{T}\mathbf{H}\mathbf{E}/2$.

Non-linear models could be easily obtained by updating $\mathbf{H}$ and its derivatives as a function of $\mathbf{E}$ at each time step. For non-uniform materials, $\mathbf{H}$ simply depends on $\bar{\mathbf{p}}$ and for anisotropic materials, $\mathbf{H}$ takes a generalized form as discussed in [Bathe 1996].

More general models can be derived using the same methodology. In appendix B, we provide formulas for volume preserving materials, and general hyperelastic materials based on the invariants of the deformation tensor.

3.5 Force and stiffness matrix

The computation of forces depends on the derivative of the strain with respect to the DOFs:

$$\boldsymbol{\nabla}_i \boldsymbol{E} = \frac{1}{2}(\mathbf{F}^\mathbf{T}\boldsymbol{\nabla}_i\mathbf{F} + \boldsymbol{\nabla}_i\mathbf{F}^T\mathbf{F}) \tag{9}$$

This $3 \times 3 \times 6$ third-order tensor is computed by differentiating equation (8) (see appendix A). We write it as a 6×6 block $\mathbf{B}_i$ using the vector form of the strain:

$$\mathbf{B}_i = \boldsymbol{\nabla}_i \mathbf{E} \tag{10}$$

The generalized force $\boldsymbol{\Gamma}_i$ acting on frame i is the negative of the energy gradient with respect to the degrees of freedom:

$$\boldsymbol{\Gamma}_i = -\int_{\mathcal{V}} \boldsymbol{\nabla}_i W^T d\mathcal{V} = -\int_{\mathcal{V}} \mathbf{B}_i^T \mathbf{H}\mathbf{E} d\mathcal{V} \tag{11}$$

The stiffness matrix $\mathbf{K}$ used in implicit integration methods (see Eq. 4) is the Jacobian of the force. Each 6×6 block $\mathbf{K}_{ij}$ encodes the variation of the force on i due to a displacement of j:

$$\mathbf{K}_{ij} = \boldsymbol{\nabla}_j \boldsymbol{\Gamma}_i = -\int_{\mathcal{V}} [\mathbf{B}_i^T \mathbf{H}\mathbf{B}_j + (\boldsymbol{\nabla}_j \mathbf{B}_i^T)\mathbf{H}\mathbf{E}] d\mathcal{V} \tag{12}$$

The first term encodes the change of intensity of the internal forces, while the second involves a third-order tensor which encodes the

change of direction. In our experiments, the latter can be safely ignored, but it may be necessary in case of large angular velocities.

3.6 External forces

Since the dynamics equation (3) is applied to the moving frames, which are the independent DOFs, each 3d external force $\mathbf{f_{ext}}$ applied to a given point $\bar{\mathbf{p}}$ must be mapped to 6d forces applied to the moving frames. Using the Jacobian and applying the power conservation law ${\mathbf{f_{ext}}}^T\dot{\mathbf{p}} = \sum_i \boldsymbol{\Gamma_{ext}}_i^T \boldsymbol{\Omega}_i$ for any $\boldsymbol{\Omega}_i$, we find that the external force is dispatched over the frames as:

$$\boldsymbol{\Gamma_{ext}}_i = \mathbf{J}_i^T(\bar{\mathbf{p}})\mathbf{f_{ext}} \tag{13}$$

4. MODELING OBJECTS

The quantities derived in section 3 are analytically defined at any point in space, except for the weights and weight derivatives that are numerically estimated as we will see in section 4.1. In this section, we address weight, sampling and integration issues.

4.1 Weight functions

Weight functions encode frame influences and must be defined at each integration point and geometry vertex. In contrast to other meshless methods, our weights do not need to constitute a partition of unity thanks to the intrinsic normalization in equation (2), and can be negative (see example with cubic splines in section 5). When normalized, they are analogous to shape functions in FEM. Moreover, by setting quasi-infinite weights near frames, our deformation field becomes interpolating and not only approximating, which is convenient to constrain and interact with the model using Dirichlet constraints. This is not the case with moving-least-squares meshless methods [Adams et al. 2008; Martin et al. 2010], where penalty terms need to be added. A frame weight function is typically a decreasing function of the distance to the frame origin in the rest configuration. A cutoff value can be set, with the constraint that each point must always be influenced by at least one frame. In figure 3, we show a U-shaped object controlled by two frames 0 and 1, and we display with colors the weight of frame 0 computed with different methods. Euclidean distance fails to capture the geometry of the object because close parts in Euclidean space move similarly (point $\bar{\mathbf{p}}$ is equally influenced by frames 0 and 1 in the example). Though geodesic distances are more accurate, artifacts can occur far from all frames. At $\bar{\mathbf{p}}$, the influence of frame 0 is not negligible compared with the influence of frame 1. As a result, this point moves along with frame 0 even if frame 1 remains fixed, contrary to what we would expect. To solve this, we use harmonic weight functions [Joshi et al. 2007] computed at initialization time. To compute the weight field of a frame, we fix its weight to a maximum value at its origin and to 0 at the origin of the others. To ease the numerical solution of the harmonic equation, our samples are aligned on a voxel grid, and we iteratively convolve the grid with a Gaussian mask until convergence. We assume that boundaries are dissipative, meaning a constant extrapolation of the weights in the Gaussian kernel. This results in the desired weight distribution as shown in the right of Figure 3 where $\bar{\mathbf{p}}$ is only influenced by frame 1. The weight gradients are required in the computation of matrix $\mathbf{W}$ involved in the strain formulation. We approximate them using central differences in the weight map. It is straightforward to rigidly attach a part of an object to a given frame, by constraining the weights of all other frames to a null value, as shown in Figure 8. This is more convenient than using high stiffnesses, which are numerically difficult. The transitions to the deformable parts are naturally obtained thanks to the smoothness of the harmonic functions. Exploring the possibility of tuning the material stiffness using the shape functions is clearly an avenue for future work. This is more difficult using FEM and traditional particle-based approaches because points do not encode rotations. [Nesme et al. 2009] encode the material stiffness within coarse elements using shape functions after a fine-level static analysis, but they use traditional shape functions and material parameters at the fine level.

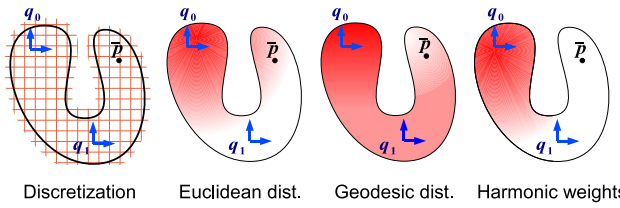

Fig. 3. Left: Discretization for volume integrals computation. Right: Weight related to frame $\mathbf{q_0}$ using different distance measures (red=max weight and white=min weight)

4.2 Volume integrals

To apply the dynamics equation (3) to a given object, we need to integrate the quantities defined in section 3 over the volume of the object. In finite elements, quadrature rules are generally available to compute the integrals in each cell as weighted sums of a small number of function values at precise parametric points. This is not possible in our meshless method. We compute the integral of any function f by regularly discretizing the volume inside the bounding box of the undeformed object:

$$\int_{\mathcal{V}} f(\bar{\mathbf{p}})d\mathcal{V} \simeq \sum_k f(\bar{\mathbf{p}}_k)\Delta v \ , \tag{14}$$

where volume Δv corresponds to the sampling resolution, and the function is null outside the object (see figure 3 left and figure 4). This straightforwardly allows us to model complex objects with non-uniform mass or stiffness, provided that a property map such as a 3d texture or a medical image is available.

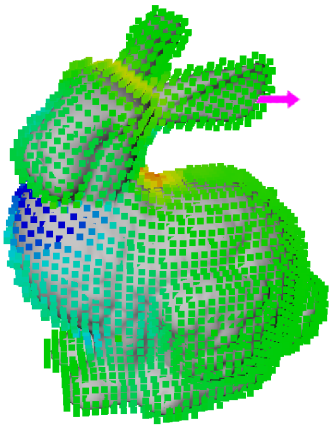

Fig. 4. Sample points in a deformed configuration with hue mapping of the strain (hi res. bunny example: 15000 samples, 2 frames)

At each time step, we compute the terms of the system's differential equation (equation (3)) by summing the local quantities defined in sections 3.2-3.6 over all the samples, as shown in Algorithm 1.

```
foreach frame i do
    Compute L̄_i ;                                          // sec. 2
    Γ_i = 0, M_i = 0;
    for frame j in 0..i do
        K_ij = K_ji = 0;
    end
end
foreach volume sample s do
    compute b'_s ;                                         // sec. 2
    compute R_s, t_s, Q_s, N_s, F_s, E_s ;                 // sec. 3.3
    foreach frame i do
        compute J_si ;                                     // eq. 7
        compute B_si, Γ_i += B_is^T H_s E_is Δv ;          // eq. 11
        M_i += ρ_s J_si^T J_si Δv ;                        // eq. 6
        for frame j in 0..i do
            K_ij -= B_is^T H_s B_js Δv ;                   // eq. 12
        end
    end
end
foreach frame i do
    for frame j in 0..i-1 do
        K_ji = K_ij^T ;
    end
end
```

Algorithm 1: Computation of forces and system matrices using the samples.

4.3 Fast pre-computed models

The computation time of the volume integrals as explained in section 4.2 is proportional to the number of volume samples. A trade-off between efficiency and speed can be obtained by tuning the resolution of the grid. It may also be possible to pre-compute a reduced set of representative voxels at initialization time in the same spirit as [An et al. 2008]. To speed up the computations even more, we show how to pre-compute the mass and stiffness matrices at initialization time. This frees us from iterating over the integration points at each time step.

Mass. For simplicity we lump the mass of each frame by neglecting the cross terms in equation (6): $\bar{\mathbf{M}}_i = \bar{\mathbf{M}}_{ii}$. The resulting global mass matrix is block diagonal, which simplifies the time integration step (equation (3)). Note that the mass has only an influence on the dynamics of the system, and not on the static solutions (i.e., equilibrium configurations), which makes the simplification hardly noticeable in many applications. We pre-compute the mass matrices in the reference configuration and we update them at each time step. Let R_i be the 3×3 rotation matrix of frame i with respect to its initial orientation and $\mathbf{R}_i$ the associated rotation matrix in 6d. The updated mass at time t is then given by rotating $\bar{\mathbf{M}}_i$ as following:

$$\mathbf{M}_i = \mathbf{R}_i \bar{\mathbf{M}}_i \mathbf{R}_i^T \quad , \quad \mathbf{R}_i = \begin{bmatrix} R_i & \mathbf{0} \\ \mathbf{0} & R_i \end{bmatrix} \tag{15}$$

Stiffness. To avoid stiffness computations at each time step, we transform the initial stiffness matrix into a generalized joint spring network.

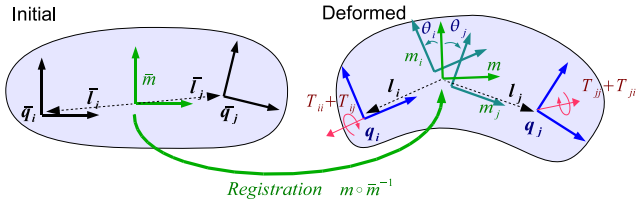

Fig. 5. Principle of the pre-computed 6d-spring system. In green: registration for stiffness warping; in black: initial frames; in blue: deformed frames

At initialization time, for each frame pair (i, j), we compute the hinge frame $\bar{\mathbf{m}}$ in the middle of the two, as illustrated in Figure 5. In a deformed configuration, the joint is represented by two frames $\mathbf{m}_i = (\mathbf{q}_i \circ \bar{\mathbf{q}}_i^{-1} \circ \bar{\mathbf{m}})$ and $\mathbf{m}_j = (\mathbf{q}_j \circ \bar{\mathbf{q}}_j^{-1} \circ \bar{\mathbf{m}})$ respectively attached to frames i and j, and the gap in between is a measure of the deformation created by the relative displacement of the moving frames. In the reference state, block $\bar{\mathbf{K}}_{ij}$ encodes the 6d force created at $\mathbf{t}_i$, the origin of frame i, as a function of a displacement of frame j expressed at $\mathbf{t}_j$,the origin of frame j. The equivalent joint stiffness $\bar{\mathbf{K}}_m$ encodes the 6d force as a function of a 6d displacement, both expressed at $\mathbf{t}_m$, the origin of frame $\mathbf{m}$. Computing the joint stiffness based on the off-diagonal block requires simple rigid body mechanics to change the points where the 6d forces and displacements are expressed. Let $\mathbf{\Theta}(j) = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{u}(j) \end{bmatrix}$ denote a small 6d displacement (velocity times unit time step) expressed at point $\mathbf{t}_j$. The same displacement expressed at point $\mathbf{t}_m$ is given by:

$$\mathbf{\Theta}(m) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{l}_{j\times} & \mathbf{I} \end{bmatrix} \mathbf{\Theta}(j) = \mathbf{D}(\mathbf{l_j})\mathbf{\Theta}(j) \ , \tag{16}$$

where $\mathbf{l}_j = (\mathbf{t}_j - \mathbf{t}_m)$, $(.)_\times$ is the cross product matrix operator. Note that $\mathbf{D}(\mathbf{a})\mathbf{D}(\mathbf{b}) = \mathbf{D}(\mathbf{a}+\mathbf{b})$ for arbitrary vectors $\mathbf{a}, \mathbf{b}$, so multiplying equation 16 with $\mathbf{D}(-\mathbf{l}_j)$ gives the inverse relation $\mathbf{\Theta}(j) = \mathbf{D}(-\mathbf{l}_j)\mathbf{\Theta}(m)$. Let $\mathbf{\Gamma}(m) = \begin{bmatrix} \boldsymbol{\tau}(m) \\ \boldsymbol{f} \end{bmatrix}$ denote a 6d force applied at point $\mathbf{t}_m$. The same force expressed at point $\mathbf{t}_i$ is given by:

$$\mathbf{\Gamma}(i) = \begin{bmatrix} \mathbf{I} & -\mathbf{l}_{i\times} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{\Gamma}(m) = \mathbf{D}(\mathbf{l}_i)^T \mathbf{\Gamma}(m) \tag{17}$$

where $\mathbf{l}_i = (\mathbf{t}_i - \mathbf{t}_m)$. Based on this, we derive the joint stiffness corresponding to an off-diagonal block as:

$$\bar{\mathbf{K}}_m = \mathbf{D}(-\bar{\mathbf{l}}_i)^T \bar{\mathbf{K}}_{ij} \mathbf{D}(-\bar{\mathbf{l}}_j) \tag{18}$$

The joint stiffness is considered constant in the joint frame $\mathbf{m}$. We project 6d vectors from this frame to the world frame using matrix $\mathbf{R}_m = \begin{bmatrix} R_m & \mathbf{0} \\ \mathbf{0} & R_m \end{bmatrix}$, where R_m is the 3×3 rotation matrix of frame $\mathbf{m}$. The joint stiffness in world coordinates is thus given by:

$$\mathbf{K}_m = \mathbf{R}_m \bar{\mathbf{K}}_m \mathbf{R}_m^T \tag{19}$$

The corresponding block of the warped stiffness matrix is derived by expressing the force applied to frame i as a function of the displacement of frame j:

$$\mathbf{K}_{ij} = \mathbf{D}(\mathbf{l}_i)^T \mathbf{K}_m \mathbf{D}(\mathbf{l}_j) \tag{20}$$

To derive the other blocks, we notice that the displacements of i and j create opposite forces because elastic forces come from relative displacements. Moreover, a joint force on one frame is associated with the opposite force on the other frame, due to Newton's third law. Based on this, and by changing the points where the displacements and forces are expressed, we derive the remaining blocks as:

$$\mathbf{K}_{ii} = -\mathbf{D}(\mathbf{l}_i)^T \mathbf{K}_m^T \mathbf{D}(\mathbf{l}_i) \quad (21)$$

$$\mathbf{K}_{jj} = -\mathbf{D}(\mathbf{l}_j)^T \mathbf{K}_m \mathbf{D}(\mathbf{l}_j) \quad (22)$$

$$\mathbf{K}_{ji} = \mathbf{D}(\mathbf{l}_j)^T \mathbf{K}_m^T \mathbf{D}(\mathbf{l}_i) \quad (23)$$

Linear and angular momentum are preserved by construction ($\mathbf{K}_{ii} = -\mathbf{D}(\mathbf{t}_i - \mathbf{t}_j)^T \mathbf{K}_{ji}$). One can easily verify that, in the reference configuration, $\mathbf{K} = \bar{\mathbf{K}}$. When we have a global rotation, $\bar{\mathbf{K}}$ is just rotated by $\mathbf{R}_m$, as expected. We can conclude that the stiffness matrix derived from our joint springs has all the expected properties.

Given frame positions, each joint force is computed based on the gap between the joint frames shown in Figure 5. Let $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ be the transformations from frames $\mathbf{m}$ to $\mathbf{m}_i$ and $\mathbf{m}$ to $\mathbf{m}_j$, respectively, expressed as 6d displacements. Since $\boldsymbol{\theta}_i = -\boldsymbol{\theta}_j$ by construction, we straightforwardly deduce the forces applied to the frames as:

$$\boldsymbol{\Gamma}_i = \mathbf{D}(\mathbf{l_i})^T (\mathbf{K}_m + \mathbf{K}_m^T) \boldsymbol{\theta}_j \quad (24)$$

$$\boldsymbol{\Gamma}_j = -\mathbf{D}(\mathbf{l_j})^T (\mathbf{K}_m + \mathbf{K}_m^T) \boldsymbol{\theta}_j \quad (25)$$

The net internal forces and stiffness matrices are computed by summing the contribution of all pairs of frames. Only the frames with intersecting weight fields interact. The weight cutoff value can be used to tune the sparsity pattern. The initial 6×6 stiffness matrices $\bar{\mathbf{K}}_m$ of these generalized joint springs are pre-computed by accumulating the contributions of sample points at arbitrarily fine resolutions. They can thus precisely encode complex material and geometry for small deformations, and they provide acceptable precision for large deformations. The location of the joint can be arbitrary. While the halfway location seems a reasonable choice, the study of its optimal position with respect to the object geometry is kept for future work.

Using this approach, the computation of the matrices using Algorithm 1 is applied only once at initialization. Then at each time step, the much simpler Algorithm 2 is applied.

```
foreach frame i do
    Γ_i = 0;
    for frame j in 0..i do
        K_ij = K_ji = 0;
    end
end
foreach frame i do
    M_i = R_i M̄_i R_i^T ;                          // sec. 4.3
    for frame j in 0..i-1 do
        add K_ij, K_ji, K_ii, K_jj ;               // eq. 20-23
        add Γ_i, Γ_j ;                             // eq. 24-25
    end
end
```

Algorithm 2: Computation of forces and system matrices using the mass-spring system.

4.4 Adaptivity

A local deformation can be modeled by inserting a frame at a desired location $\mathbf{p}^*$ within the deformed object, as shown in Figure 9. To insert a frame, we need to estimate the associated undeformed position $\bar{\mathbf{p}}(\mathbf{p}^*)$. The analytical inversion of the skinning mapping function $\mathbf{p}(\bar{\mathbf{p}})$ is difficult. Fortunately, we can efficiently solve the equation numerically using Newton's method and the deformation gradient $\mathbf{F} = \partial \mathbf{p} / \partial \bar{\mathbf{p}}$ of equation (8). The initial guess $\bar{\mathbf{p}}^{(0)}$ is set to the closest sample point. We then iteratively refine the solution using: $\bar{\mathbf{p}}^{(k+1)} = \bar{\mathbf{p}}^{(k)} + \mathbf{F}(\bar{\mathbf{p}}^{(k)})^{-1}(\mathbf{p}^* - \mathbf{p}(\bar{\mathbf{p}}^{(k)}))$. The search typically converges in 3 iterations with a stop criterion of $\|\mathbf{p}^* - \mathbf{p}(\bar{\mathbf{p}}^{(k)})\| < 10^{-5}$.

Once the additional frame is inserted, we compute its weight field in the reference state and update the mass-spring network accordingly. This adds a new mass matrix block along with a number of new stiffness blocks. Moreover, all previous mass and stiffness blocks whose spatial domain intersects the inserted weight field must be recomputed. To speed up these computations, the weights of the inserted frame are based on geodesic distances, quickly evaluated in a single propagation pass through the voxel grid up to the desired cutoff value. Each voxel contains the list of frames and weights which influence it, and contributes to the (lumped) mass block using equation (6) and to new stiffness blocks using equation (12).

5. VALIDATION AND RESULTS

5.1 Implementation

Our method has been integrated in the simulation platform SOFA (`http://www.sofa-framework.org/`) and will be freely available in the upcoming release. In principle, any time integration scheme can be used to solve equation (3), although we prefer the implicit Euler scheme with a conjugate gradient solver for stability and large time-steps. For implicit schemes, the stiffness matrix $\mathbf{K}$ is evaluated to provide force variations given a displacement. The solver returns an updated generalized velocity $\boldsymbol{\Omega}_i = [\boldsymbol{\omega}_i^T \; \dot{\mathbf{t}}_i^T]^T$ for each frame i.

In our framework, we decouple physics, collision handling and visualization through three different models. Any type of visual and collision models can be plugged into our method (e.g., discrete/parametric surface/volume, bounding volume hierarchy, etc.) as we provide a volumetric displacement field, as well as mapping functions $\mathbf{J}_i$ to transfer interaction forces to the frames.

The main simulation loop maps samples to their deformed positions and computes the strain, the mapping function and the strain derivative at their location. For visual model points, only deformed positions are required, while for collision points, the mapping function $\mathbf{J}_i$ is also needed. To handle collisions and self-collisions, the fast GPU-based detection method presented in [Faure et al. 2008] and the method for collision response presented in [Allard et al. 2010] are used. Finally, we accumulate, for each frame, sample contributions in terms of mass, force and stiffness (space integration), handle collisions and perform time integration.

For the fast pre-computed version of the simulation (section 4.3), there is no physical sample involved, since the material behavior is encoded in the pre-computed stiffness matrices.

5.2 Accuracy

To validate the mechanical accuracy of our method, we model the deformation of a beam using regularly spaced frames along the axis, with piecewise linear weight functions. We apply an

extension force to the beam and verify that the force-extension law precisely matches the theoretical St. Venant-Kirchhoff model $f = \epsilon + 3\epsilon^2/2 + \epsilon/2$, independently of the number of frames and the volume sample densities. Bending is more complex because it simultaneously involves extension-compression and shear, especially with large displacements as shown in the example in Figure 6. We obtain similar results with FEM, our model being slightly less extended, as illustrated in the left of the figure. This confirms that accurate continuum mechanics can be performed using our model. In the right of Figure 6, we show bars deformed

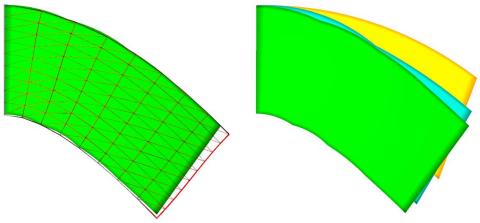

Fig. 6. Left: comparison of our model (green) with 5 frames along the main axis and $10 \times 10 \times 9$ samples in the volume, with a St-Venant-Kirchhoff model (red) with $10 \times 10 \times 9$ points, with same loads and material parameters. Right: comparison of beams with 5, 3 and 2 frames in our model, from front to back.

using different numbers of frames. As usual, fewer degrees of freedom result in more stiffness. As with FEM, the weight functions have an important influence on the results. The slight bumps on the interpolated surface are due to the nearly constant curvature between two consecutive frames, resulting from the dual quaternion blending between the two frames. A Catmull-Rom spline interpolation can generate a smoother shape due to the negative weights that couples non-consecutive frames. However, the gradient of the weights is not constant, producing inhomogeneous deformations which is not what we expect with homogeneous materials.

5.3 Deformation modeling and adaptivity

The most appealing feature of our method is probably its ability to easily model deformable objects using a reduced number of control primitives. In the video, the T-shaped rubber object shown in Figure 7 (Young's modulus $E = 200kPa$, Poisson's ratio $\nu = 0.3$) exhibits compression, shear, bending and torsion using only two frames, corresponding to a total of 12 DOF. The same number of DOF only allows to model a single linear tetrahedron in FEM, which can not exhibit torsion and bending! In the example with three frames, the mechanical properties are computed by integration over the volume as described in section 4. The object automatically exhibits an asymmetric stiffness reflecting its asymmetric shape.

The turtle shown in Figure 8 ($E = 700kPa$, $\nu = 0.3$) illustrates the ability to tune the stiffness using the shape functions. The turtle is animated using seven frames, and the weights associated with the frame located in the shell appear as color maps. In the middle, the weights are computed using standard harmonic coordinates. The shell is not rigid because the weight of the shell frame is lower in the regions near the neck and near the legs, which thus undergo non-rigid motion blending. In the right of the figure, we have imposed a high weight inside the whole shell and let the weight in the remaining volume be computed automatically. This results in a rigid shell, as shown in the video.

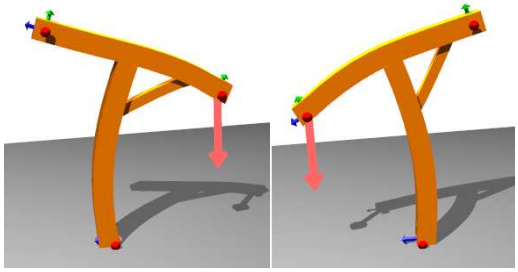

Fig. 7. An object with asymmetric stiffness automatically computed based on its asymmetric shape.

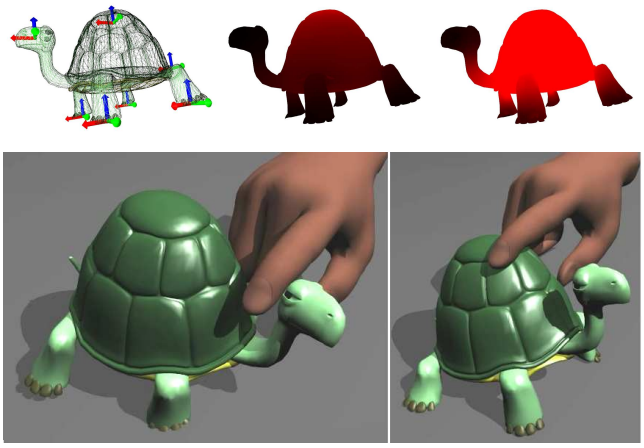

Fig. 8. Modeling stiffness using weight functions. Top: (Left) The deformation frames. (Middle) Shell weights automatically computed. (Right) Weights set to model a rigid shell. Bottom: the turtle with a rigid shell.

In the example shown in Figure 1, two frames are sufficient to model a deformable bunny ($E = 700kPa$, $\nu = 0.3$) . Adding flexibility to an ear is done using an additional frame, which is dramatically simpler than editing the mesh of an FEM model. In the same video, we show that a dynamically inserted frame at the contact point with an object can be used to generate a local deformation, as illustrated in Figure 9. The range of the local deformation can be tuned using the weight function of the inserted frame. Such a high level of adaptivity in a physical model is straightforward with our model, while it is difficult to implement using previous methods.

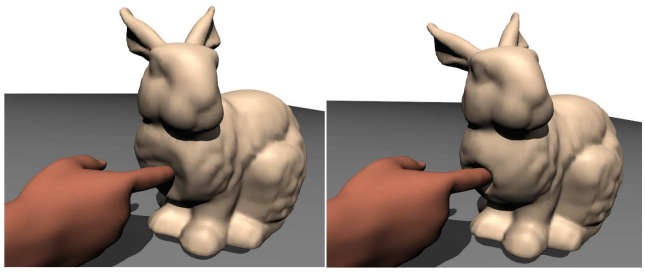

Fig. 9. More or less global deformation produced by dynamically inserting a frame with two different weight functions

5.4 Performance

The FEM simulation of the bar in Figure 6 runs at 65 fps. While our model with 5 frames runs at 38 fps, the speed increases dramatically to 700 fps using precomputed generalized springs. For a static solution implemented using Newton's method, the convergence to the equilibrium takes 20 times fewer iterations than using FEM, due to a lower number of DOFs.

Table I summarizes the simulation performance for the turtle and bunny models on a $2.33GHz$ quad-core PC without parallel implementation. The integration of one time step, shown in the column labeled "physical time", takes an average of $7ms$. We have tested two resolutions for the computation of volume integrals, yielding minor changes in the deformation. One is suited for interactive simulation while the other is more adapted to off-line rendering. Both are robust to large deformations. Further speedup can be obtained with the fast spring network presented in Section 4.3 whose complexity only depends on the number of interacting frames. However, the overall computation time also depends on the number of vertices of the visual and collision models. As expected, the spring model provides good results in relatively large linear deformations, but is not as robust for large non-linear deformations. This issue will be investigated in future work.

Table I. Summary of simulation performance

Model	**Visual** time (ms) (# points)	**Collision** time (ms) (# points)	**Physical** time (ms) (# points/frame)	Precomp. time (s)	fps
Turtle (low res.)	4 (6642)	40 (6642)	106 (2313 / 7)	5	7
Turtle (hi res.)	4 (6642)	40 (6642)	648 (15452 / 7)	110	1.4
Turtle (springs)	4 (6642)	40 (6642)	10 (0 / 7)	110	19
Bunny (low res.)	10 (34000)	7 (2500)	51 (2323 / 2)	1.5	15
Bunny (hi res.)	10 (34000)	7 (2500)	250 (15526 / 2)	34	3.7
Bunny (springs)	10 (34000)	7 (2500)	10 (0 / 2)	34	37

The results reported in Table I were obtained using our current implementation, with a lot of room for future optimization. The speed greatly depends on the number of integration points, as shown by the comparison between low resolution and high resolution. At each conjugate gradient iteration used in our ODE solver, the stiffness matrix product requires the mapping of frame displacements to strain changes, then the computation of the corresponding stress changes, and finally the mapping back to frame forces (see eq. 12). The intensive part of this computation is the product with the matrix $\mathbf{B}$, which currently is implemented as a dense matrix even though it is sparse. Another avenue for optimization is to further reduce the number of integration points, which is currently huge compared with the number of independent DOF, even in the low resolution version.

At each iteration, our implicit solver not only computes the force changes due to internal forces, but also due to contact forces. Frame displacements are thus also mapped to displacements of the vertices of the collision model and contact forces are mapped back to the frames. Here also, our implementation does not exploit sparsity; moreover, a further optimization could skip vertices without contact forces. This unnecessary processing of the collision vertices explains why the computation time is higher than expected when using precomputed generalized springs. The model shown in Figure 6 has no collisions and allows an easier comparison with FEM. As few as 5 frames (30 independent DOFs) allow us to obtain a smooth bent shape similar to the shape obtained using $10 \times 10 \times 9$ FEM nodes (2700 DOFs). Since our method assumes a locally uniform strain around each integration point, we found that $10 \times 10 \times 9$ integration points were required to obtain the same amount of bending using the same material parameters. We can quantify the difference between two methods by measuring the distance between the endpoints of the centerline, relative to the length of the bar. In this configuration, the difference between FEM and our method is only 2.5%. When the number of integration points are the same in the two methods, the computation time per iteration should be similar, provided that the same level of optimization is applied to each implementation. However, the number of iterations to converge depends on the number of independent DOFs. In this regard our method is significantly better, needing 5 times fewer iterations, as can be seen in the accompanying video.

We also compared the methods at coarser resolutions, as shown in Figure 10. We used the same number of FEM nodes as integration points in the frame-based method, as with the previous comparison. Three frames were used with $2 \times 2 \times 2$ points, and five frames with $4 \times 4 \times 4$ points. The frame-based model becomes softer as the resolution decreases, while the FEM model becomes stiffer. We notice that the FEM converges faster than our method on the coarse example. However, our method is closer to the reference result. In the $4 \times 4 \times 4$ example, the distance between the endpoint of the centerline in our model from the endpoint in the reference FEM is 13%, while it is 25% in the FEM. In the $2 \times 2 \times 2$ example, the distance is 15% in our model, while it is 42% in the FEM.

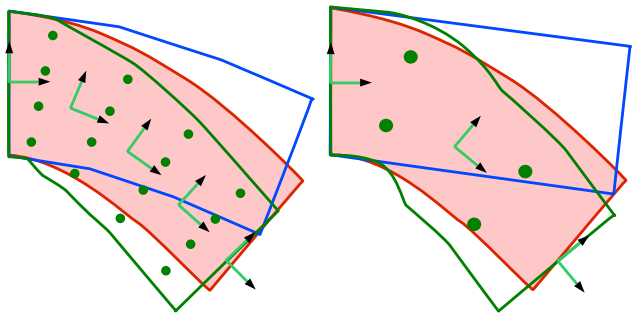

Fig. 10. Comparison of FEM simulation (in blue) and frame-based simulation (in green) for $4 \times 4 \times 4$ (left) and $2 \times 2 \times 2$ (right) FEM nodes and integration points. The reference high-resolution FEM is shown in red. The integration points are shown as disks. The green curve is the outline of a visual model bound to the frames.

Our method clearly outperforms the FEM when fine meshes are required to preserve smoothness while the spatial deformation frequencies are sufficiently low so that it can be captured by a reduced number of DOFs. It can be seen as a new type of reduced model, more local and more intuitive than the reduced models produced by traditional modal analysis.

6. CONCLUSION

We have presented a new type of deformable model using continuum mechanics applied to objects undergoing skinning deformation fields. Our approach allows the creation of sparse meshless models with arbitrary constitutive laws, and we have demonstrated it using St. Venant-Kirchhoff materials. The models are robust to large displacements and deformations. We have shown that

the behavior of objects with complex materials and geometries can be simulated using a small number of moving frames. Moreover, the deformation can be encoded in a computationally efficient way using six-dimensional mass-spring systems precomputed from the mass and stiffness matrices. Compared with FEM, adaptivity is easier because no volumetric mesh is used, and mixing rigid and deformable frames is straightforward using appropriate shape functions. Sampling is easier than with traditional particle-based meshless methods because there is no constraint on the number and on the placement of the frames. Moreover, the ability to use constrained weight fields, rather than radial basis functions, simplifies the design and simulation of objects composed of rigid and deformable parts.

Currently, the range of the local deformations obtained by inserting frames at contact points depends on the distance to the other frames. This limitation will be addressed in future work. More generally, the optimal placement of frames and the design of weight functions will have to be investigated, as well as automatic insertion and deletion of frames to handle large deformations or even topological changes. Hardware implementations would certainly accelerate the mapping of the visual and collision models, the computation of the volume integrals and allow better on-the-fly resampling. The relation between stiffness and weight functions could be exploited such as in [Nesme et al. 2009] to simulate fine distributions of heterogeneous materials using coarse frame sampling. Another interesting study will be the comparison, in the context of physically-based simulation, of various blending methods developed in geometric modeling such as the linear combination of matrices (i.e., linear blend skinning) or the linear combination of the log-quaternion representations [Weber et al. 2007].

ACKNOWLEDGMENTS

This work is funded in part by the French National Research Agency (ANR), the Canadian Institutes of Health Research, Canada Research Chairs Program, NSERC, Peter Wall Institute for Advanced Studies, MITACS, and European project “Passport for Liver Surgery” (FP7, ICT-2007.5.3). We would like to thank Florent Falipou and Michaël Adam for preparing the examples and the video.

REFERENCES

ADAMS, B., OVSJANIKOV, M., WAND, M., SEIDEL, H.-P., AND GUIBAS, L. 2008. Meshless modeling of deformable shapes and their motion. In Symposium on Computer Animation. 77–86.

ALLARD, J., FAURE, F., COURTECUISSE, H., FALIPOU, F., DURIEZ, C., AND KRY, P. 2010. Volume contact constraints at arbitrary resolution. ACM Transactions on Graphics 29, 3.

AN, S. S., KIM, T., AND JAMES, D. L. 2008. Optimizing cubature for efficient integration of subspace deformations. ACM Trans. Graph. 27, 5, 1–10.

BARAFF, D. AND WITKIN, A. 1998. Large steps in cloth simulation. SIGGRAPH Comput. Graph. 32, 106–117.

BARBIČ, J. AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. ACM Transactions on Graphics (SIGGRAPH 2005) 24, 3 (Aug.), 982–990.

BATHE, K. 1996. Finite Element Procedures. Prentice Hall.

COTIN, S., DELINGETTE, H., AND AYACHE, N. 1999. Real-time elastic deformations of soft tissues for surgery simulation. IEEE TVCG 5, 62–73.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. 2001. Dynamic real-time deformations using space and time adaptive sampling. SIGGRAPH Comput. Graph., 31–36.

FAURE, F., BARBIER, S., ALLARD, J., AND FALIPOU, F. 2008. Image-based collision detection and response between arbitrary volumetric objects. In ACM Siggraph/Eurographics Symposium on Computer Animation.

FRIES, T.-P. AND MATTHIES, H. 2003. Classification and overview of meshfree methods. Tech. rep., TU Brunswick, Germany.

GOURRET, J.-P., THALMANN, N. M., AND THALMANN, D. 1989. Simulation of object and human skin formations in a grasping task. SIGGRAPH Comput. Graph. 23, 3, 21–30.

GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. Charms: A simple framework for adaptive simulation. SIGGRAPH Comput. Graph., 281–290.

GROSS, M. AND PFISTER, H. 2007. Point-Based Graphics. Morgan Kaufmann.

HAN, D.-P., WEI, Q., AND LI, Z.-X. 2008. Kinematic control of free rigid bodies using dual quaternions. International Journal of Automation and Computing 5, 3, 319–324.

IRVING, G., TERAN, J., AND FEDKIW, R. 2006. Tetrahedral and hexahedral invertible finite elements. Graph. Models 68, 2.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. ACM Trans. Graph. 26, 3, 71.

KAUFMANN, P., MARTIN, S., BOTSCH, M., AND GROSS, M. 2008. Flexible simulation of deformable models using discontinuous galerkin fem. In Symposium on Computer Animation.

KAVAN, COLLINS, ZARA, AND O’SULLIVAN. 2007. Skinning with dual quaternions. In Symposium on Interactive 3D graphics and games. 39–46.

KAVAN, L., COLLINS, S., ZARA, J., AND O’SULLIVAN, C. 2008. Geometric skinning with approximate dual quaternion blending. ACM Trans. Graph. 27, 4.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint dependent local deformations for hand animation and object grasping. In Graphics interface. 26–33.

MARTIN, S., KAUFMANN, P., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2010. Unified simulation of elastic rods, shells, and solids. SIGGRAPH Comput. Graph. 29, 3.

MARTIN, S., KAUFMANN, P., BOTSCH, M., WICKE, M., AND GROSS, M. 2008. Polyhedral finite elements using harmonic basis functions. Comput. Graph. Forum 27, 5.

MÜLLER, M. AND GROSS, M. 2004. Interactive virtual materials. In Graphics Interface.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. ACM Transactions on Graphics 24, 3, 471–478.

MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In Symposium on Computer Animation. 141–151.

NADLER, B. AND RUBIN, M. 2003. A new 3-d finite element for nonlinear elasticity using the theory of a cosserat point. Int. J. of Solids and Struct. 40, 4585–4614.

NEALEN, A., MÜLLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2005. Physically based deformable models in computer graphics. In Comput. Graph. Forum. Vol. 25 (4). 809–836.

NESME, M., KRY, P., JERABKOVA, L., AND FAURE, F. 2009. Preserving Topology and Elasticity for Embedded Deformable Models. SIGGRAPH Comput. Graph..

O'BRIEN, J. AND HODGINS, J. 1999. Graphical models and animation of brittle fracture. SIGGRAPH Comput. Graph., 137–146.

PAI, D. K. 2002. STRANDS: Interactive simulation of thin solids using Cosserat models. Computer Graphics Forum, The International Journal of the Eurographics Association 21, 3, 347–352.

PLATT, S. AND BADLER, N. 1981. Animating facial expressions. SIGGRAPH Comput. Graph., 245–252.

SIFAKIS, E., DER, K. G., AND FEDKIW, R. 2007. Arbitrary cutting of deformable tetrahedralized objects. In Symposium on Computer Animation.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. In SIGGRAPH Comput. Graph., M. C. Stone, Ed. Vol. 21. 205–214.

TERZOPOULOS, D. AND QIN, H. 1994. Dynamic nurbs with geometric constraints for interactive sculpting. ACM Trans. Graph. 13, 2, 103–136.

TESCHNER, M., HEIDELBERGER, B., MULLER, M., AND GROSS, M. 2004. A versatile and robust model for geometrically complex deformable solids. In CGI.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. In Comp. Graph. Forum (Proc. of Eurographics). Vol. 26.

WEISS, J., MAKER, B., AND GOVINDJEE, S. 1996. Finite element implementation of incompressible, transversely isotropic hyperelasticity. Comp. Meth. in App. Mech. and Eng. 135, 107–128.

APPENDIX

A. DETAILED MATRIX EXPRESSIONS

Dual quaternion corresponding to the current frame position:

$$\mathbf{q}_i = \begin{bmatrix} \mathbf{q}_0^i \\ \mathbf{q}_\varepsilon^i \end{bmatrix} = [a_0^i\, b_0^i\, c_0^i\, w_0^i\, a_\varepsilon^i\, b_\varepsilon^i\, c_\varepsilon^i\, w_\varepsilon^i]^T$$

where $\mathbf{q}_0^i$ is the real part representing rotations and $\mathbf{q}_\varepsilon^{iT} = (\mathbf{t}_{i\wedge} \circ \mathbf{q}_0^i)/2$ is the dual part representing translations to the frame origin $\mathbf{t}_i = [t_x^i\, t_y^i\, t_z^i]^T$ and $\mathbf{t}_{i\wedge} = [t_x^i\, t_y^i\, t_z^i\, 0]^T$.
Relative transformations from the rest configuration:

$$\mathbf{q}_i \circ \bar{\mathbf{q}}_i^{-1} = \bar{\mathbf{T}}_i \mathbf{q}_i \;\;,\;\; \bar{\mathbf{T}}_i = \begin{bmatrix} H^-[(\bar{\mathbf{q}}_0^i)^{-1}] & \mathbf{0} \\ H^-[(\bar{\mathbf{q}}_\varepsilon^i)^{-1}] & H^-[(\bar{\mathbf{q}}_0^i)^{-1}] \end{bmatrix}$$

$$H^-(\mathbf{q}) = \begin{bmatrix} w & c & -b & a \\ -c & w & a & b \\ b & -a & w & c \\ -a & -b & -c & w \end{bmatrix} \;\;,\;\; \mathbf{q} = \begin{bmatrix} a \\ b \\ c \\ w \end{bmatrix}$$

where H^- is the Hamilton operator that encodes right quaternion products.
The linearly blended dual quaternion is:

$$\mathbf{b} = \sum w_i \bar{\mathbf{T}}_i \mathbf{q}_i = \begin{bmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_\varepsilon \end{bmatrix}$$

Normalized dual quaternion:

$$\begin{aligned} \mathbf{b}' = \frac{\mathbf{b}}{\|\mathbf{b}\|} &= \frac{1}{\|\mathbf{Q}_0\|} \begin{bmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_\varepsilon \end{bmatrix} - \frac{\mathbf{Q}_\varepsilon.\mathbf{Q}_0}{\|\mathbf{Q}_0\|^3} \begin{bmatrix} \mathbf{0} \\ \mathbf{Q}_0 \end{bmatrix} \\ &= [\mathbf{q}_0^T\, \mathbf{q}_\varepsilon^T]^T = [a_0\, b_0\, c_0\, w_0\, a_\varepsilon\, b_\varepsilon\, c_\varepsilon\, w_\varepsilon]^T \end{aligned}$$

The rigid transformation equivalent to $\mathbf{b}'$, that maps reference material point positions to current positions through frame displacements, is [Kavan et al. 2008]:

$$\mathbf{R}(\mathbf{b}') = \begin{bmatrix} 1-2b_0^2-2c_0^2 & 2a_0b_0-2w_0c_0 & 2a_0c_0+2w_0b_0 \\ 2a_0b_0+2w_0c_0 & 1-2a_0^2-2c_0^2 & 2b_0c_0-2w_0a_0 \\ 2a_0c_0-2w_0b_0 & 2b_0c_0+2w_0a_0 & 1-2a_0^2-2b_0^2 \end{bmatrix}$$

$$\text{and} \quad \mathbf{t}(\mathbf{b}') = 2 \begin{bmatrix} -w_\varepsilon a_0 + a_\varepsilon w_0 - b_\varepsilon c_0 + c_\varepsilon b_0 \\ -w_\varepsilon b_0 + a_\varepsilon c_0 + b_\varepsilon w_0 - c_\varepsilon a_0 \\ -w_\varepsilon c_0 - a_\varepsilon b_0 + b_\varepsilon a_0 + c_\varepsilon w_0 \end{bmatrix}$$

There is a linear relationship between velocity and rate of the frame: $\dot{\mathbf{q}}_i = (\hat{\boldsymbol{\omega}}_i \circ \mathbf{q}_i)/2$ [Han et al. 2008]. It can be represented by a 8×6 matrix $\mathbf{L}_i$ such as $\dot{\mathbf{q}}_i = \mathbf{L}_i \boldsymbol{\Omega}_i$ and we have:

$$\mathbf{L}_i(\mathbf{q}_i) = \boldsymbol{\nabla}_i \mathbf{q}_i = \frac{1}{2} \begin{bmatrix} \mathbf{L}_{i0} & \mathbf{0} \\ \mathbf{L}_{i\varepsilon} & \mathbf{L}_{i0} \end{bmatrix} \;\;,\;\; \mathbf{L}_{i0} = \begin{bmatrix} w_0^i & c_0^i & -b_0^i \\ -c_0^i & w_0^i & a_0^i \\ b_0^i & -a_0^i & w_0^i \\ -a_0^i & -b_0^i & -c_0^i \end{bmatrix}$$

$$\mathbf{L}_{i\varepsilon} = \begin{bmatrix} w_\varepsilon^i + c_0^i t_z^i + b_0^i t_y^i & c_\varepsilon^i - w_0^i t_z^i - b_0^i t_x^i & -b_\varepsilon^i + w_0^i t_y^i - c_0^i t_x^i \\ -c_\varepsilon^i + w_0^i t_z^i - a_0^i t_y^i & w_\varepsilon^i + c_0^i t_z^i + a_0^i t_x^i & a_\varepsilon^i - w_0^i t_x^i - c_0^i t_y^i \\ b_\varepsilon^i - w_0^i t_y^i - a_0^i t_z^i & -a_\varepsilon^i + w_0^i t_x^i - b_0^i t_z^i & w_\varepsilon^i + b_0^i t_y^i + a_0^i t_x^i \\ -a_\varepsilon^i - b_0^i t_z^i + c_0^i t_y^i & -b_\varepsilon^i + a_0^i t_z^i - c_0^i t_x^i & -c_\varepsilon^i - a_0^i t_y^i + b_0^i t_x^i \end{bmatrix}$$

To compute the Jacobian (equation (7)) and the deformation gradient (equation (8)), we need the first-order derivatives $\mathbf{Q}$, $\mathbf{N}$ and $\mathbf{W}$ that are given by (see also the long equation (26)) :

$$\mathbf{N}(\mathbf{b}, \mathbf{b}') = \frac{\partial \mathbf{b}'}{\partial \mathbf{b}} = \begin{bmatrix} \mathbf{N}_0 & \mathbf{0} \\ \mathbf{N}_\varepsilon & \mathbf{N}_0 \end{bmatrix}$$

$$\text{with} \;\; \mathbf{N}_0 = -\frac{1}{\|\mathbf{Q}_0\|}(\mathbf{q}_0 \mathbf{q}_0^T - \mathbf{I})$$

$$\text{and} \;\; \mathbf{N}_\varepsilon = -\frac{1}{\|\mathbf{Q}_0\|}(\mathbf{q}_0 \mathbf{q}_\varepsilon^T + \mathbf{q}_\varepsilon \mathbf{q}_0^T) - \frac{\mathbf{Q}_\varepsilon.\mathbf{Q}_0}{\|\mathbf{Q}_0\|^3}(\mathbf{I} - \mathbf{q}_0 \mathbf{q}_0^T)$$

$$\mathbf{W}(\bar{\mathbf{p}}) = \frac{\partial \mathbf{b}}{\partial \bar{\mathbf{p}}} = \sum \bar{\mathbf{T}}_i \mathbf{q}_i \mathbf{W}_i \quad \text{with} \quad \mathbf{W}_i = \frac{\partial w_i}{\partial \bar{\mathbf{p}}}$$

$\mathbf{W_i}$ represents the weight gradients and are numerically computed in the voxel grid by central differences.
The derivative of the deformation gradient is needed to compute the derivative of the strain (equation (9)). Applying the derivation chain rule and using Einstein summation convention, we obtain a third-order tensor:

$$\begin{aligned} \boldsymbol{\nabla}_i \mathbf{F}_{abc} &= w_i [\tfrac{\partial \mathbf{R}}{\partial \mathbf{b}'}_{abd} \mathbf{N}_{de} \bar{\mathbf{T}}_{i\,ef} \mathbf{L}_{i\,fc}] + \mathbf{Q}_{ad} \mathbf{N}_{de} [\tfrac{\partial \mathbf{W}}{\partial \mathbf{q}_i}_{ebf} \mathbf{L}_{i\,fc}] \\ &+ w_i [\tfrac{\partial \mathbf{Q}}{\partial \mathbf{b}'}_{ade} \mathbf{N}_{ef} \bar{\mathbf{T}}_{i\,fg} \mathbf{L}_{i\,gc}] \mathbf{N}_{dh} \mathbf{W}_{hb} + w_i \mathbf{Q}_{ad} [\tfrac{\partial \mathbf{N}}{\partial \mathbf{b}}_{def} \bar{\mathbf{T}}_{i\,fg} \mathbf{L}_{i\,gc}] \mathbf{W}_{eb} \end{aligned}$$

Notations can be simplified by expressing each 3×3 submatrix $\boldsymbol{\nabla}_i \mathbf{F}_k = \partial \mathbf{F} / \partial \boldsymbol{\Omega}_{ik}$ using the operators $\Delta\mathbf{R}$, $\Delta\mathbf{W}$, $\Delta\mathbf{Q}$ and $\Delta\mathbf{N}$:

$$\begin{aligned} \boldsymbol{\nabla}_i \mathbf{F}_k &= w_i \Delta\mathbf{R}([\mathbf{N}\bar{\mathbf{T}}_i \mathbf{L}_i]^k) + \mathbf{Q}\mathbf{N}\bar{\mathbf{T}}_i \Delta\mathbf{W}([\mathbf{L}_i]^k) \\ &+ w_i \Delta\mathbf{Q}([\mathbf{N}\bar{\mathbf{T}}_i \mathbf{L}_i]^k)\mathbf{N}\mathbf{W} + w_i \mathbf{Q} \Delta\mathbf{N}([\bar{\mathbf{T}}_i \mathbf{L}_i]^k)\mathbf{W} \end{aligned}$$

Each operator produces a matrix from a 8×1 column vector $\mathbf{V} = [\mathbf{V}_0^T\, \mathbf{V}_\varepsilon^T]^T = [v_0\, v_1\, v_2\, v_3\, v_4\, v_5\, v_6\, v_7]^T$ (for instance the k^{th} column of $\mathbf{N}\bar{\mathbf{T}}_i \mathbf{L}_i$). They are obtained by differentiating $\mathbf{R}$, $\mathbf{W}$, $\mathbf{Q}$ and $\mathbf{N}$ (see also long equations (27) and (28)):

$$\Delta\mathbf{W}(\mathbf{V}) = \mathbf{V}\mathbf{W}_i$$

$$\mathbf{Q}(\bar{\mathbf{p}},\mathbf{b}') = \frac{\partial \bar{\mathbf{p}}}{\partial \mathbf{b}'} = 2\begin{bmatrix} -w_\varepsilon + a_0x + b_0y + c_0z & c_\varepsilon - b_0x + a_0y + w_0z & -b_\varepsilon - c_0x - w_0y + a_0z & a_\varepsilon + w_0x - c_0y + b_0z & w_0 & -c_0 & b_0 & -a_0 \\ -c_\varepsilon + b_0x - a_0y - w_0z & -w_\varepsilon + a_0x + b_0y + c_0z & a_\varepsilon + w_0x - c_0y + b_0z & b_\varepsilon + c_0x + w_0y - a_0z & c_0 & w_0 & -a_0 & -b_0 \\ b_\varepsilon + c_0x + w_0y - a_0z & -a_\varepsilon - w_0x + c_0y - b_0z & -w_\varepsilon + a_0x + b_0y + c_0z & c_\varepsilon - b_0x + a_0y + w_0z & -b_0 & a_0 & w_0 & -c_0 \end{bmatrix} \quad (26)$$

$$\Delta\mathbf{R}(\mathbf{V}) = 2\begin{bmatrix} -2(b_0v_1 + c_0v_2) & -c_0v_3 + b_0v_0 + a_0v_1 - w_0v_2 & b_0v_3 + c_0v_0 + w_0v_1 + a_0v_2 \\ c_0v_3 + b_0v_0 + a_0v_1 + w_0v_2 & -2(a_0v_0 + c_0v_2) & -a_0v_3 - w_0v_0 + c_0v_1 + b_0v_2 \\ -b_0v_3 + c_0v_0 - w_0v_1 + a_0v_2 & a_0v_3 + w_0v_0 + c_0v_1 + b_0v_2 & -2(a_0v_0 + b_0v_1) \end{bmatrix} \quad (27)$$

$$\Delta\mathbf{Q}(\mathbf{V}) = 2\begin{bmatrix} xv_0 + yv_1 + zv_2 - v_7 & zv_3 + yv_0 - xv_1 + v_6 & -yv_3 + zv_0 - xv_2 - v_5 & xv_3 + zv_1 - yv_2 + v_4 & v_3 & -v_2 & v_1 & -v_0 \\ -zv_3 - yv_0 + xv_1 - v_6 & xv_0 + yv_1 + zv_2 - v_7 & xv_3 + zv_1 - yv_2 + v_4 & yv_3 - zv_0 + xv_2 + v_5 & v_2 & v_3 & -v_0 & -v_1 \\ yv_3 - zv_0 + xv_2 + v_5 & -xv_3 - zv_1 + yv_2 - v_4 & xv_0 + yv_1 + zv_2 - v_7 & zv_3 + yv_0 - xv_1 + v_6 & -v_1 & v_0 & v_3 & -v_2 \end{bmatrix} \quad (28)$$

$$\Delta\mathbf{N}(\mathbf{V}) = \begin{bmatrix} \Delta\mathbf{N}_0 & \mathbf{0} \\ \Delta\mathbf{N}_\varepsilon + \Delta\mathbf{N}_c & \Delta\mathbf{N}_0 \end{bmatrix} \quad \text{with:}$$

$$\Delta\mathbf{N}_0 = -\frac{1}{\|\mathbf{Q}_0\|^2}[\mathbf{q}_0\mathbf{V}_0^T + \mathbf{V}_0\mathbf{q}_0^T + \mathbf{V}_0.\mathbf{q}_0(\mathbf{I} - 3\mathbf{q}_0\mathbf{q}_0^T)]$$

$$\Delta\mathbf{N}_\varepsilon = -\frac{1}{\|\mathbf{Q}_0\|^2}[\mathbf{q}_0\mathbf{V}_\varepsilon^T + \mathbf{V}_\varepsilon\mathbf{q}_0^T + \mathbf{V}_\varepsilon.\mathbf{q}_0(\mathbf{I} - 3\mathbf{q}_0\mathbf{q}_0^T)]$$

$$\Delta\mathbf{N}_c = -\frac{1}{\|\mathbf{Q}_0\|^2}[\mathbf{q}_\varepsilon\mathbf{V}_0^T + \mathbf{V}_0\mathbf{q}_\varepsilon^T + \mathbf{V}_0.\mathbf{q}_\varepsilon(\mathbf{I} - 3\mathbf{q}_0\mathbf{q}_0^T) - 3\mathbf{V}_0.\mathbf{q}_0(\mathbf{q}_\varepsilon\mathbf{q}_0^T + \mathbf{q}_0\mathbf{q}_\varepsilon^T)] - 2\frac{\mathbf{Q}_\varepsilon.\mathbf{Q}_0}{\|\mathbf{Q}_0\|^2}\Delta\mathbf{N}_0$$

B. EXTENSION TO OTHER MATERIALS

Volume preservation: For volume preserving materials, we need to compute the unit volume change with respect to the degrees of freedom. The unit volume is the determinant of the deformation gradient: $J = det(\mathbf{F})$. Its derivative is obtained from already defined matrices:

$$\boldsymbol{\nabla}_i J_k = \frac{\partial J}{\partial \boldsymbol{\Omega}_{ik}} = det(\mathbf{F}) \sum_{lj} \mathbf{F^{-T}}_{lj} \boldsymbol{\nabla}_i \mathbf{F}_{ljk}$$

Inverted elements have a null strain and strain energy, resulting in a stable but undesired configuration. To circumvent this, an energy term with bulk modulus k is generally added to penalize volume change from the initial configuration. Possible expressions for the energy density, force and stiffness are:

$$W = \frac{k}{2}(J-1)^2 \ , \ \boldsymbol{\Gamma}_i = -k(J-1)\boldsymbol{\nabla}_i J^T \ , \ \mathbf{K}_{ij} \approx -k\boldsymbol{\nabla}_i J^T \boldsymbol{\nabla}_j J$$

Isotropic hyperelastic material: a widely spread approach in mechanical engineering is to express the energy density W in terms of the three invariants of the right Cauchy-Green deformation tensor $\mathbf{C} = \mathbf{F^T F}$. They are function of the strain and volume:

$$\begin{aligned} I1(\mathbf{C}) &= Tr(\mathbf{C}) = 2\varepsilon_{xx} + 2\varepsilon_{yy} + 2\varepsilon_{zz} + 3 \\ I2(\mathbf{C}) &= (Tr(\mathbf{C})^2 - Tr(\mathbf{C^2}))/2 \\ &= 4(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz} + \varepsilon_{xx}\varepsilon_{yy} + \varepsilon_{xx}\varepsilon_{zz} \\ &\quad + \varepsilon_{yy}\varepsilon_{zz} - \varepsilon_{xy}^2 + \varepsilon_{xz}^2 + \varepsilon_{yz}^2) + 3 \\ I3(\mathbf{C}) &= det(\mathbf{C}) = J^2 \end{aligned}$$

Their derivatives are obtained using the rows of $\mathbf{B}$:

$$\begin{aligned} \boldsymbol{\nabla}_i I1 &= 2(\mathbf{B}_{i1} + \mathbf{B}_{i2} + \mathbf{B}_{i3}) \\ \boldsymbol{\nabla}_i I2 &= 2\boldsymbol{\nabla}_i I1 - 8(\varepsilon_{xy}\mathbf{B}_{i4} + \varepsilon_{xz}\mathbf{B}_{i5} + \varepsilon_{yz}\mathbf{B}_{i6}) + 4(\varepsilon_{xx} \\ &\quad (\mathbf{B}_{i2} + \mathbf{B}_{i3}) + \varepsilon_{yy}(\mathbf{B}_{i1} + \mathbf{B}_{i3}) + \varepsilon_{zz}(\mathbf{B}_{i1} + \mathbf{B}_{i2})) \\ \boldsymbol{\nabla}_i I3 &= 2J\boldsymbol{\nabla}_i J \end{aligned}$$

The relationship $\partial W/\partial I_i$ is given by the material model (e.g., Mooney-Rivlin). Plugging this relationship into the following equations, we can compute forces and stiffness matrices as:

$$\boldsymbol{\Gamma}_i = -[\frac{\partial W}{\partial I1}\boldsymbol{\nabla}_i I1 + \frac{\partial W}{\partial I2}\boldsymbol{\nabla}_i I2 + \frac{\partial W}{\partial I3}\boldsymbol{\nabla}_i I3]^T$$

$$\mathbf{K}_{ij} \approx -[\boldsymbol{\nabla}_i I1^T \boldsymbol{\nabla}_j \frac{\partial W}{\partial I1} + \boldsymbol{\nabla}_i I2^T \boldsymbol{\nabla}_j \frac{\partial W}{\partial I2} + \boldsymbol{\nabla}_i I3^T \boldsymbol{\nabla}_j \frac{\partial W}{\partial I3}]$$

For incompressible materials, one can use reduced invariants instead. They are obtained using previously introduced variables:

$$\begin{aligned} \bar{I1} &= J^{-2/3} I1 \\ \bar{I2} &= J^{-4/3} I2 \\ \bar{I3} &= J \end{aligned} \quad (29)$$